AN ARTIFICIAL INTELLIGENCE APPROACH TO THE BEHAVIORAL MODELING OF ASYNCHRONOUS SEQUENTIAL LOGIC CIRCUITS

Sheng-Fu Wu and P. David Fisher
Department of Electrical Engineering, Michigan State University
East Lansing, MI 48824-1226
phone (517) 355-5241

Abstract

A method has been developed which uses artificial intelligence techniques to model the functional behavior of asynchronous sequential logic circuits (ASLCs). It provides a highly structured, interactive approach. The domain representation, production rules, and control strategy are described. The *Behavioral Descriptor* of the ASLC design automation system generates a primitive flow table which captures the ASLC's functional behavior. This methodology has been implemented in the C programming language and currently resides on Sun workstations. This ASLC design automation system has been validated using numerous representative examples and has been found to be much quicker and more reliable than more traditional methods for designing ASLCs.

Introduction

The fast growth in VLSI technology enhances the importance of logic design. Asynchronous sequential logic circuits (ASLCs) play a very important role in the design of logic circuits. An ASLC state machine is potentially faster than its clocked sequential logic circuit (CSLC) counterpart since it does not have to wait for the arrival of a clock pulse before causing a state transition [1-8]. However, for a given state machine, the ASLC design process is much more complex and time consuming than that of the CSLC. This drawback has led state machine designers to prefer CSLC architectures to their ASLC counterparts. Consequently, before the intrinsic advantages of ASLC architectures can be fully exploited, there exists a need for efficient ASLC design tools which significantly simplify the process of designing this class of sequential logic functions.

The application of Artificial Intelligence (AI) to digital system design has been widely studied by many researchers [9-14], and many AI applications in engineering are concentrated on VLSI design [15]. We describe an artificial intelligence (AI) approach to the behavioral modeling of ASLCs. A "behavioral descriptor" (***BD***) is built to describe the circuit's behavior with a primitive flow table (PFT). The ***BD*** accurately generates a PFT in a very short period of time, so that it lessens the burden on circuit designers and reduces the design cycle time. This feature becomes more important when large-scale ASLCs must be designed.

We have developed and tested an ASLC design automation system which includes the ***BD***. Such an ASLC design system is illustrated in Fig. 1. It consists of five modules, each of which can accept data files from either an up-stream module or interactively from the circuit designer [16]. These modules perform the following functions:

(1) ***Behavioral Descriptor***: maps the functional design specifications into a primitive flow table (PFT), which completely captures the ASLC's behavioral model.

(2) ***Merger***: reduces the PFT into a merged flow table, thereby minimizing the number of states by eliminating redundant primitive state assignments.

(3) ***Connector***: converts the merged flow table into an adjacency table, adds states as needed to avoid critical races.

(4) ***Assigner***: encodes the states and generates the state excitation table and corresponding output table from the information stored in the adjacency table, merged flow table, and previously developed output table.

(5) ***Equation Generator***: eliminates static hazards and converts the excitation table and output table into state equations and output equations, which are placed in two-level sum-of-products form.

Figure 1 provides a view of the design hierarchy traversed by the ASLC design system. The highest level concerning ASLC design system is the behavioral level. At this level the input/output behavior of an ASLC state machine is described. The work reported in this paper deals primarily with this level.

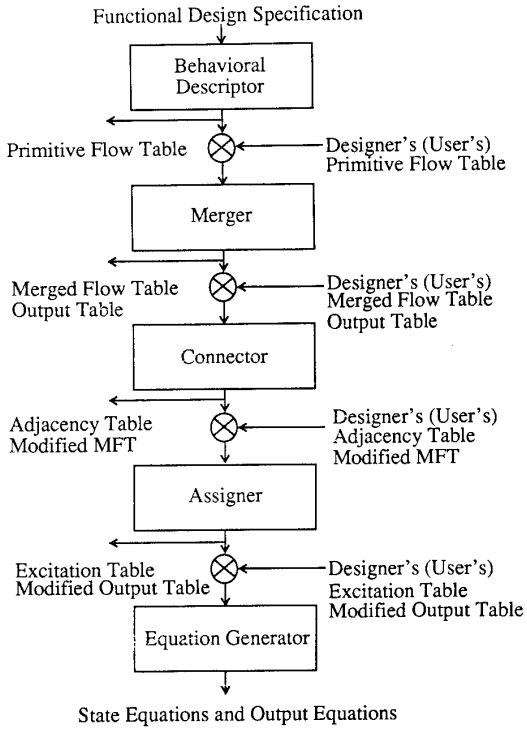

Fig. 1. Basic architecture of the ASLC design system

CH2819-1/90/0000-1115$01.00©1991 IEEE

VISIT…

Domain Problem Solver -- The Behavioral Descriptor

Developing the behavioral model is the first important step in designing an ASLC. The inputs of this domain problem are the inputs, outputs, and function of the desired ASLC, and these must be provided by the designer. The output (result) of this domain problem is a primitive flow table which describes the behavior of the desired ASLC of the designer. Therefore, the main idea is to build a problem solver, which is named the "behavioral descriptor" (***BD***). The ***BD*** will map the designer's design specification into a PFT.

The ASLC behavioral model includes two basic parameters: ASLC inputs and outputs. Each primitive state corresponds to a unique allowed input/output combination. Four important parts in the BD are described:

(1) ***Design Specification***: three high level specifications; namely, the input specification, the output specification, and the functional specification.

(2) ***Facts***: states represented by the input/output combinations and the corresponding state numbers which are generated by the control strategy.

(3) ***Production Rules***: "IF conditions THEN actions" clauses. There are no built-in production rules for ***BD***. All the production rules are generated according to the design specification.

(4) ***Control Strategy***: controls state transitions in the PFT and generates the next PFT entries. They determine which production rule is to be fired by matching each pattern in the condition part of a rule with the changing pattern in the input signals. The control procedure looks for the facts and fills the state number into the corresponding entry in the PFT until all the entries have been filled with the assigned values.

Representation

According to Rich [17], it is often useful to divide the representation question into three sub-questions:

(1) How can individual objects and facts be represented?

(2) How can the representations of individual objects be combined to form a representation of a complete problem state?

(3) How can the sequences of problem states that arise in a search process be represented efficiently?

Our specific criteria was to choose a representation that allows all of the necessary knowledge to be represented and facilitates its use in solving the problem at hand. Moreover, we wanted a representation that would be simple, informative, and easily used interactively on a computer monitor (workstation). Our specific implementation is as follows:

*Input Representation for the **BD***: The inputs to the ***BD*** is the design specification of the ASLC. The format for describing the design specification is described as follows.

The input specification has the following format:
input_specification
in_name_1 in_name_2 in_name_3 ... in_name_n

The output specification has the following format:
output_specification
out_name_1 out_name_2 out_name_3 ... out_name_m

The sequential logic functions's functional behavior is completely specified by the designer and has the following format:
function_specification
IF in_name_1 state in_name_2 state...
THEN out_name_1=function_1 out_name_2=function_2...

*Internal Representation of Facts for **BD*** Following is the definition for the attributes of the entry in the PFT.
row_no: specifies the entry's row.
column_no: specifies the entry's column.
state_no: specifies the entry's primitive state number.
stable: specifies whether the entry is in stable state or not.
output_value: specifies the value of the output signal combination associated with a primitive state.

Production Rules

Many AI applications employ some form of if-then rule-based programming; i.e., if conditions C_1, C_2,..., C_n happen, then actions A_1, A_2,..., A_m will be performed. Most of the VLSI design tools have built-in production rules which are fixed [10-12]. MYCIN provides production rules which can be modified [17]. However, there is no built-in production rules for ***BD***. All the production rules are generated according to the design specification. Therefore, it is a dynamic structure.

Control Strategy

The control strategy manipulates the representation by matching the production rules. Each time it looks for one signal changing in the input state of the ASLC and searches for the matched production rule. If a match occurs, it takes the appropriate action and generates the next output signal. At this time, the combination of input signals and output signals forms a new primitive state. The control procedure decides whether this state currently exists. If it is a new state, the control procedure assigns it a new value. Next, the control procedure places the value in the corresponding row and column in the PFT. This process continues until all rows and columns have been filled with the assigned primitive state values or "don't cares" values, where "don't cares" indicate that the particular primitive state transition does not occur.

Design Example and Discussion

The ***BD*** has been executed on many examples. Some examples have quite large PFTs, which cannot be shown in this paper. In order to explain how the ***BD*** works, we use a simple example to illustrate it. This example deals with a specific sequential logic function (SLF) illustrated in Fig. 2(a). This logic element, referred to as a P-SLF, is a multi-stable sequential logic function and has the following set of *Functional Design Specifications*:

(1) There are two input logic lines, which are labeled A and B.

(2) There are two output logic lines. They are labeled D or d and E or e, where the upper-case and lower-case letters are used to denote next-output and current-output state, respectively.

(3) The output logic lines change state only on the falling edge of B, and the next output states are defined as follows: D = eA; E = d + A.

The ***Behavioral Descriptor*** *maps the Functional Design Specification* into the *Primitive Flow Table*. The *Functional Design Specification* must first be read from a file or entered interactively by the circuit designer. The interactive approach would be as follows:

1. Enter the sequential logic function's input variable names.

input_specification
in_name_1 in_name_2 in_name_3 ... in_name_n
A B

2. Enter the sequential logic function's output variable names.

ouput_specification
out_name_1 out_name_2 out_name_3 ... out_name_m
D E

3. Enter the sequential logic function's functional behavior, which describes the relationship between the next output state and the present output state and present and past input states.

function_specification
IF in_name_1 state in_name_2 state...
IF B 1-to-0
THEN out_name_1=function_1 out_name_2 =function_2...
THEN D=E&A E=D+A.

Once the *Functional Design Specification* has been entered into the computer, the ***BD*** generates primitive state transition rules. For example, if B in the above example makes a transition from 1-to-0, then the next output state is updated as indicated above; otherwise, there is no change.

After these rules have been generated, the primitive flow table (PFT) is constructed and completed. If the sequential logic function has m input lines and n output lines, then 2^{m+n} primitive states are possible with each of these primitive states being assigned a row in the PFT. There would be 2^m possible distinct input states with each of these states being assigned a unique column in the PFT. For the P-SLF, m = 2 and n = 2; so, the PFT would be expected to contain up to sixteen rows and four columns.

The PFT is initially empty. The process used to complete the PFT is as follows: One of the allowed I/O combinations is arbitrarily selected as the starting primitive state and marked stable to initiate the process of filling the PFT. For the P-SLF example, ABDE = 0000 satisfies the Functional Design Specification and represents a stable primitive state; so, this primitive state can be selected as a starting point. From this starting point, the logic state of one input line is changed and the rules applied to compute the "next primitive state". All allowed input transitions are explored. Not all combinations of the input and output logic states may be allowed by the Functional Design Specification. For example, for the P-SLF, ABDE = 1010 is not allowed since E = d + A. Moreover, only those primitive state transitions are allowed for which one and only one input logic line changes state. For example, for the P-SLF, ABDE = 0000 ⇒ 1101 is not allowed because input logic lines A and B were changed simultaneously. The PFT is completed after each allowed primitive state has been visited and its row in the PFT completed.

Once the *Functional Design Specification* has been entered (or read), the ***BD*** automatically generates the PFT (see Fig. 2(b)). For the P-SLF, twelve primitive states are delineated with stable primitive states being "marked". (For illustrative purposes, we have circled the stable states in Fig. 2(b)). From the PFT, the circuit designer can readily identify all of the primitive state assignments. For example, primitive state 6 is defined as having present-input and present-output states of BA = 11 and DE = 01, respectively. From this figure we also see that the ASLC design system imposed the fundamental-mode rule on the input states; i.e., only one input logic line may change state at a time. The circuit designer may use this PFT output from the ***BD*** module to provide documentation of the design or to verify manually the correctness of the result. But the latter is not generally necessary since the ***BD*** in the ASLC design system automatically verifies the correctness of the PFT.

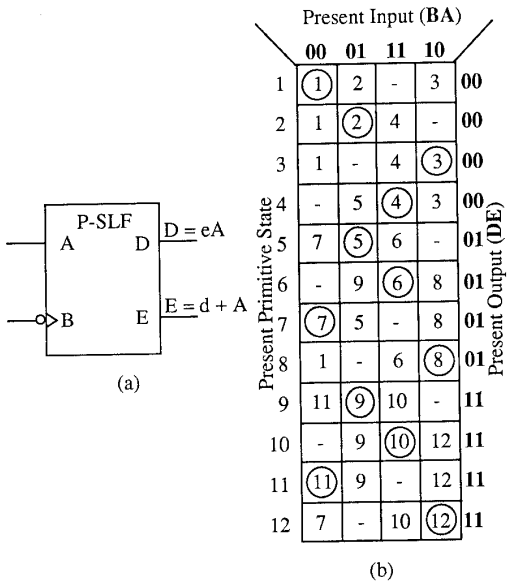

Fig. 2. P-SLF (a) graphic symbol and (b) primitive flow table.

Conclusions

Design is the process of refining a representation at one level of abstraction to a more detailed representation at a lower level of abstraction. The design of an ASLC can be a very difficult and time consuming process because of the large number of inputs/outputs, hazards, and critical races. Many inputs/outputs can make flow tables unmanageably large. The "behavioral descriptor", referred to here as simply ***BD***, overcomes these complexity problems. Its general usefulness has been demonstrated by applying it to a wide range of representative ASLC design problems.

Once the PFT has been generated by the ***BD***, the complete behavior of the designed ASLC is controllable and observable. Since we can easily predict the next outputs from the current state and the next inputs by using BD, there is no need to draw the timing diagram for knowing the ASLC behavior. The BD helps us quickly produce the PFT without using pen-and-paper methods. It provides the information about the circuit behavior and speedup the ASLC design process.

Acknowledgments

The authors wish to thank Mr. Jun-Woo Kang for his assistance in validating the ASLC design system described here and for his many thoughtful comments and suggestions.

References

[1] I. E. Sutherland, "Micropipelines," *Commun. of the ACM*, vol. 32, pp. 720-738, June 1989.

[2] A. E. A. Almaini, *Electronic Logic Systems*, 2nd ed., Prentice-Hall International (UK) Ltd., 1989.

[3] McCluskey, E. J., *Logical Design Principles*, Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[4] L. A. Hollaar, "Direct Implementation of Asynchronous Control Units," *IEEE Trans. Computers*, vol. C-31, pp. 1133-1141, Dec. 1982.

[5] K. J. Breeding, *Digital Design Fundamentals*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[6] A. D. Friedman, *Fundamentals of Logic Design and Switching Theory*, Computer Science Press, Inc., Rockville, Md.,1986.

[7] S. H. Unger, *Asynchronous Sequential Circuits*, Wiley Interscience, New York, 1969.

[8] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1978.

[9] V. E. Kelly, "The Critter Systems: Automated Critiquing of Digital Circuit Designs," *Proceedings of the 21st ACM/IEEE Design Automation Conference*, pp. 419-425, IEEE and ACM-SIGDA, IEEE Computer Society, June 1984.

[10] R. Joobbani, *An Artificial Intelligence Approach to VLSI Routing*, Kluwer Academic Publishers, Boston/Dordrecht/Lancaster, 1986.

[11] R. Joobbani and D. P. Siewiorek, "WEAVER: A Knowledge-Based Routing Expert," *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, pp. 266-272, IEEE and ACM-SIGDA, IEEE Computer Society, June 23-26, 1985.

[12] W. P. Bermingham, and J. H. Kim, "DAS/Logic: A Rule-based Logic Design Assistant," *Proceedings of the Second Conference, Miami Beach, Florida, Artificial Intelligence Applications The Engineering of Knowledge-Based Systems*, pp. 264-268, IEEE Computer Society Press, Dec. 1985.

[13] W. P. Birmingham, D. P. Siewiorek, "MICON: A Knowledge Based Single Board Computer Designer," *Proceedings of the 21st ACM/IEEE Design Automation Conference*, pp. 565-571, IEEE and ACM-SIGDA, IEEE Computer Society, June 25-27, 1985.

[14] L. I. Steinberg, T. M. Mitchell, "A Knowledge-based Approach to VLSI CAD," *Proceedings of the 21st ACM/IEEE Design Automation Conference*, pp. 412-418, IEEE and ACM-SIGDA, IEEE Computer Society, June 25-27, 1985.

[15] W. B. Rauch_Hindin, *Artificial Intelligence in Business, Science, and Industry*, Prentice Hall, Englewood Cliffs, New Jersey, 1985.

[16] S. F. Wu and P. D. Fisher, "Automating the Design of Asynchronous Sequential Logic Circuits," *IEEE 1990 Custom Integrated Circuits Conference*, pp. 29.5.1 - 29.5.4, May 1990.

[17] E. Rich, *Artificial Intelligence*, McGraw-Hill, Inc., New York, 1983.